The Evolution of Trees and Logistic Regression

Gerardo De la O

2022

Tree Methods

In this section, we will delve into Tree-based algorithms. Tree based algorithms hold some key advantages over other methods due to their non-parametric nature. Some benefits include:

1. Take continuous and categorical variables as-is (no need to one-hot encode, scale, normalize, etc.)
2. Ignore redundant variables (not sensitive to multicollinearity)
3. Able to generate non-linear fits without constraining the output function

Of course, the benefits come at the cost of interpretability and, in the case of simple trees, performance. In general, however, we can solve the performance issue with more "evolved" tree methods. The relative performance of Tree methods goes as follows:

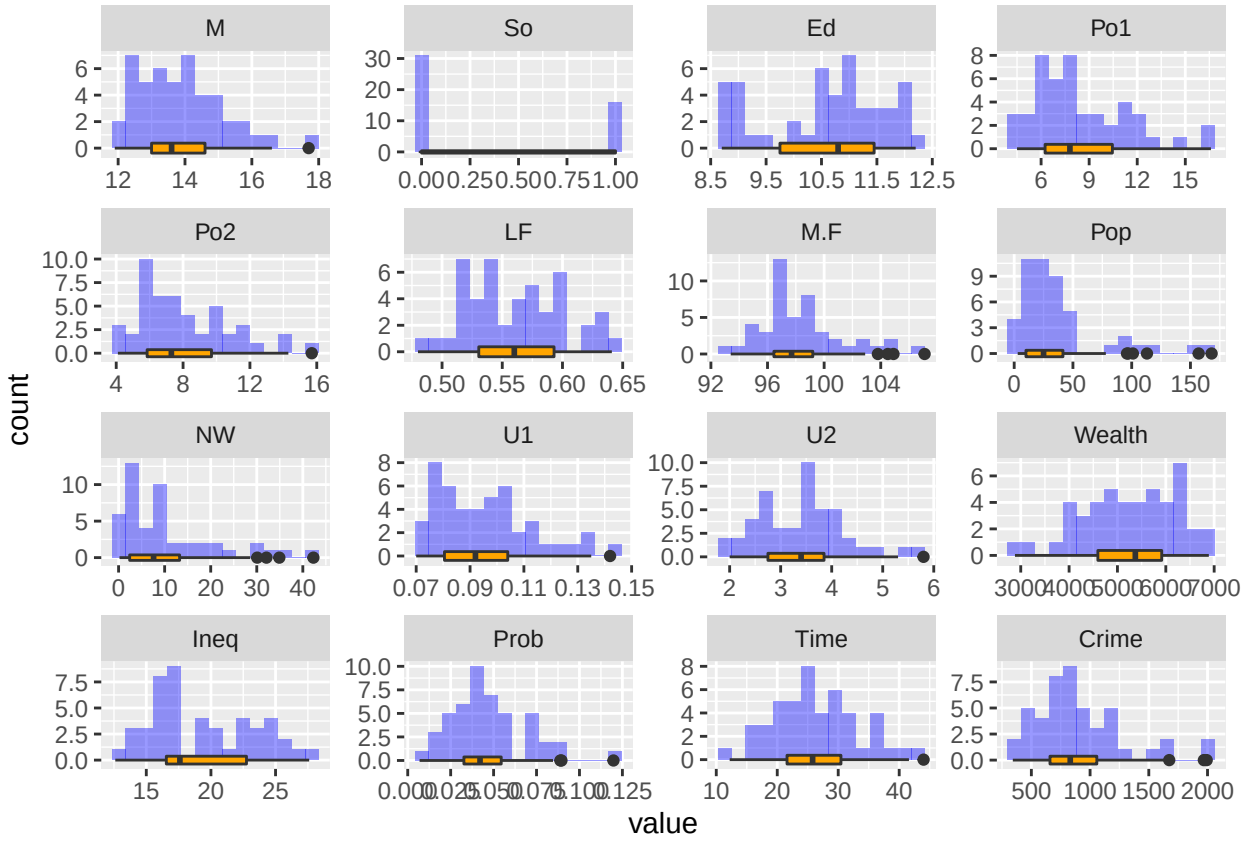
single tree < bagged trees < random forest < boosted trees

Our goal for this exercise is to attempt to find a good regression tree model and a good random forest model for the crime data we've been using in the past few assignments.

Preparation

The purpose of this step is to understand the data and perform some basic cleanup. Removing outliers is the only transformation we applied here.

```
library(tidyverse)
library(reshape2)
library(lubridate)
library(caret)

#import data
filename = 'http://www.statsci.org/data/general/uscrime.txt'
data <- read.csv(filename, sep = '\t' ) #tab delimited


# Melt to plot easily
melted_data <- melt(data)

ggplot(data=melted_data, aes(x = value)) +
    geom_histogram(bins=15, fill='blue', alpha=0.4) +
    geom_boxplot(fill='orange') +
    facet_wrap(~variable, scales = "free")
```




```
# Outlier removal
z_scores <- as.data.frame(sapply(data, function(data) (abs(data-mean(data))/sd(data))))
delete = rowSums(z_scores > 3)

df <- data[!delete, ]
dim(df)
```

```
## [1] 42 16
```

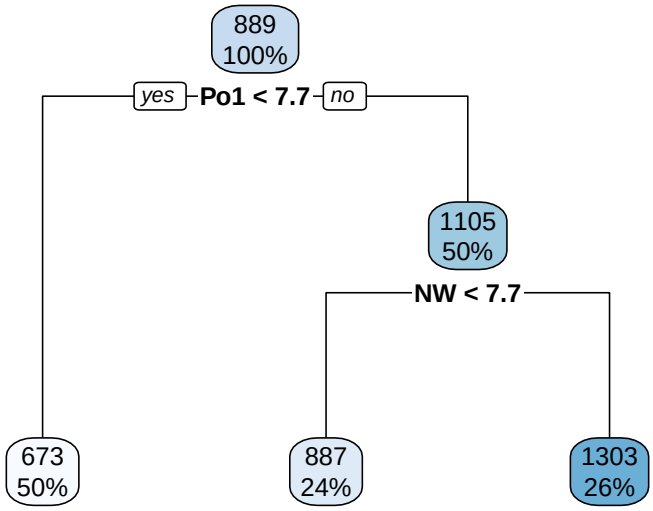
a) Regression Tree Model

Regression trees work by essentially partitioning the data into subgroups, then fitting a simple constant for each observation in the subgroup (usually the average). Here, we first fit a single regression tree using the 'rpart' method within the caret package. The best model from this method had an RMSE value of 323.3 using 5-fold cross validation. Since we are fitting a single tree, we can visualize it directly. It turns out Po1 and NW are the only variables needed for the tree to make a good prediction!

```
library(rpart.plot)

set.seed(1)

cv <- trainControl(method = "cv",  number = 5)

# CV single regression tree
t1 <- train(Crime ~ ., data = df, method = "rpart", trControl = cv)

print(t1)
```

```
## CART
##
## 42 samples
## 15 predictors
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 34, 34, 34, 33, 33
## Resampling results across tuning parameters:
##
##   cp          RMSE      Rsquared   MAE
##   0.06783675  325.3229  0.2975217  255.1643
##   0.16466898  346.1152  0.2182910  266.8073
##   0.35610206  345.0240  0.2182910  267.1139
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was cp = 0.06783675.
```

```
rpart.plot(t1$finalModel)
```



The question then becomes, can we do better? And the answer is yes. Bagging (Bootstrap Aggregating) fits many large trees to re-sampled versions of our training data, then classifies by majority vote. In theory, this produces a smoother decision boundary and reduces overfitting because many models are "averaged out". In fact, from the results, we can see an RMSE value of 297.5, so bagging did improve our model!

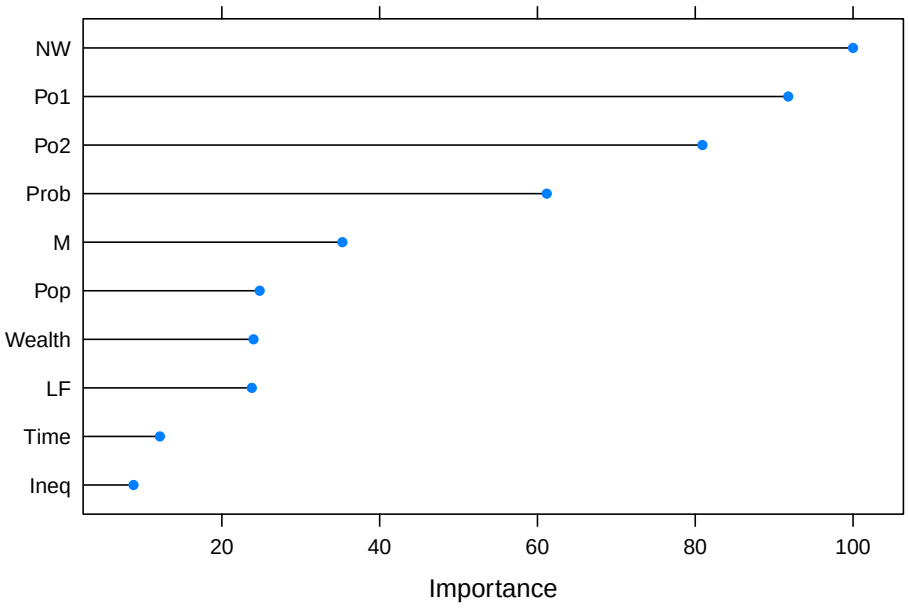
Since bagging uses multiple trees, its not easy to visualize. However, we can plot the relative importance of each variable on the overall model. From the plot, it is clear that NW and Po1 are the most important variables, which is consistent with our single tree (although the order is flipped).

```
set.seed(1)

cv <- trainControl(method = "cv",  number = 5)
```

```r
# CV bagged CART
t2 <- train(Crime ~ ., data = df, method = "treebag", trControl = cv, importance = TRUE)

t2_model = t2$finalModel

print(t2)
```

```
## Bagged CART
##
## 42 samples
## 15 predictors
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 34, 34, 34, 33, 33
## Resampling results:
##
##   RMSE     Rsquared   MAE
##   297.503  0.5330564  224.5819
```

```r
plot(varImp(t2), 10)
```



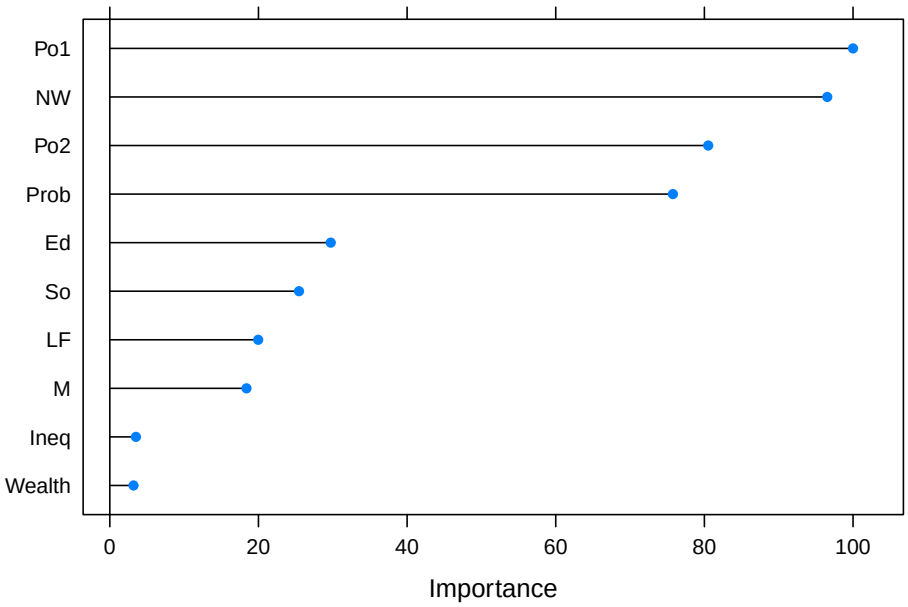
b) Random Forest Model

Finally, we'll see if we can do even better with random forests. Random forests employ a similar concept to bagging, but the idea is to have the sampled trees as uncorrelated as possible. This is accomplished by sampling on features on top of the bootstrap sample. In theory, random forests should be more accurate than single or bagged trees, and are relatively hard to overfit. From the results, we can see an RMSE value of 297.1, so Random forests had very marginal improvement over our bagged tree, nonetheless it is the best model we've built so far!

```r
set.seed(1)

cv <- trainControl(method = "cv",  number = 5)

# CV random forest
t3 <- train(Crime ~ ., data = df, method = "rf", trControl = cv, importance = TRUE)

t3_model = t3$finalModel

print(t3)
```

```
## Random Forest
##
## 42 samples
## 15 predictors
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 34, 34, 34, 33, 33
## Resampling results across tuning parameters:
##
##   mtry  RMSE      Rsquared   MAE
##    2    300.2198  0.5170121  220.9474
##    8    297.0973  0.5446002  214.7447
##   15    304.8027  0.5422921  222.8384
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was mtry = 8.
```

```r
plot(varImp(t3), 10)
```



Real World Example - Logistic Regression

I currently work at a B2B technology company specializing in conversational analytics. From a a business perspective, Logistic Regression can be used to help our client companies gain insight into what drives certain key metrics such as sales, churn, Net Promoter Scores, etc. An impactful use case of logit models, for example, would be to predict whether a customer will provide a high or a low "Customer Satisfaction" survey response.

Predictors that I would consider are:

1. Reason for call (What the customer said they're calling about)
2. Indicators of negative experience (repeat interactions, frustration, confusion, etc.)
3. Positive agent behaviors (advocacy, acknowledgement, issue resolution, etc.)
4. Negative agent behaviors (powerless to help language, transfers, long holds, etc.)
5. Customer metadata (tenure, past purchases, previous survey responses, etc.)

On top of being able to measure the probability of a customer providing a low score, we are able to use the predictor coefficients to find the relative effect of specific events against the outcome. This is particularly important because some of these predictors can be controlled by the company! The model essentially gives you the things you should investigate first, assuming the goal is to minimize low survey scores.

Logistic Regression for Credit Risk

This section explores the use of Logistic Regression to classify whether credit applicants are good or bad credit risks.

Preparation

The first thing to note is that Logit models are parametric. As such, they cannot take in categorical variables without manipulation. Luckily, the data source provides a numeric data set just for this purpose, so we use that for this model. Here, we also split the data set into training and test sets to be able to fairly evaluate performance.

```
library(ROCR)
set.seed(1)

f = 'http://archive.ics.uci.edu/ml/machine-learning-databases/statlog/german/german.data-numeric'
data <- read.table(f)
data$V25 <-ifelse(data$V25==1,0,1)

# Split into Train/Test sets
smp_size <- floor(.8 * nrow(data))
sampler <- sample(seq_len(nrow(data)), size = smp_size)
train <- data[sampler, ]
test <- data[-sampler, ]
```

Model and Results

Now we are ready to fit the model and analyze some results! The code below shows all the details, but things of note are summarized here:

1. The model is predicting "Bad" credit risk (likelihood of a bad application)
2. Since the cost of False Negatives (incorrectly predict "good") is 5x higher than the cost of False Positives, we lean the model towards bad classifications. The threshold that balances cost was 22% for this particular model.
3. After adjustment, the accuracy of the model is 70%, with a 5-to-1 ratio of False Positives to False negatives.
4. We include a ROC curve, indicating the model is much better than random at predicting bad scores.

```r
set.seed(1)
#Run Logistic Model
model <- glm( V25 ~ ., family=binomial(link='logit'),data=train)

print(summary(model))
```

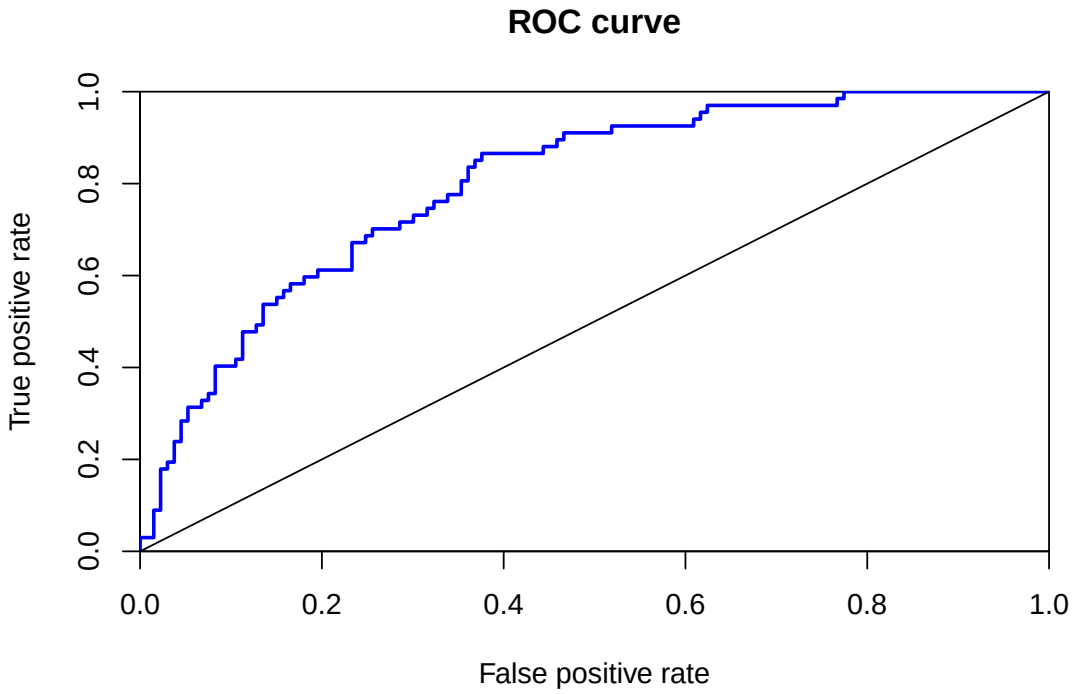
```
##
## Call:
## glm(formula = V25 ~ ., family = binomial(link = "logit"), data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.0760  -0.6994  -0.4046   0.7475   2.6083
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  3.773672   1.356103   2.783 0.005390 **
## V1          -0.590381   0.080186  -7.363  1.8e-13 ***
## V2           0.031617   0.009729   3.250 0.001155 **
## V3          -0.304998   0.096702  -3.154 0.001610 **
## V4           0.004250   0.004392   0.968 0.333149
## V5          -0.223191   0.066601  -3.351 0.000805 ***
## V6          -0.181212   0.085797  -2.112 0.034678 *
## V7          -0.270628   0.128622  -2.104 0.035373 *
## V8           0.007651   0.094091   0.081 0.935193
## V9           0.215423   0.113403   1.900 0.057482 .
## V10         -0.015354   0.009640  -1.593 0.111196
## V11         -0.372090   0.123785  -3.006 0.002648 **
## V12          0.233829   0.178266   1.312 0.189626
## V13          0.096962   0.266611   0.364 0.716094
## V14         -0.206687   0.210086  -0.984 0.325204
## V15         -1.523092   0.693096  -2.198 0.027983 *
## V16          0.576499   0.218112   2.643 0.008214 **
## V17         -0.896575   0.391325  -2.291 0.021956 *
## V18          0.868749   0.469190   1.852 0.064084 .
## V19          0.924739   0.641011   1.443 0.149126
## V20          0.134330   0.404966   0.332 0.740111
## V21         -0.345249   0.357880  -0.965 0.334692
## V22         -0.737264   0.695750  -1.060 0.289296
## V23          0.167377   0.365100   0.458 0.646636
## V24          0.125396   0.294696   0.426 0.670465
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 965.23  on 799  degrees of freedom
## Residual deviance: 735.63  on 775  degrees of freedom
## AIC: 785.63
##
## Number of Fisher Scoring iterations: 5
```

```r
# Result Analysis
test$prob <- predict(model, test, type = "response")
```

```r
# Want to minimize FNs, so we need to make it "easier" to classify as bad (lower threshold)
thres =.22 #this balances the cost of FPs and FNs!
test_results <- test %>%
    mutate(pred = 1*(prob > thres), accurate = 1*(pred == V25)) %>%
    select(V25, prob, pred, accurate)

# Confusion Matrix
confusion <- confusionMatrix(
    data = as.factor(test_results$pred),
    reference = as.factor(test_results$V25))
confusion
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 83 10
##          1 50 57
##
##                Accuracy : 0.7
##                  95% CI : (0.6314, 0.7626)
##     No Information Rate : 0.665
##     P-Value [Acc > NIR] : 0.1652
##
##                   Kappa : 0.4135
##
##  Mcnemar's Test P-Value : 4.782e-07
##
##             Sensitivity : 0.6241
##             Specificity : 0.8507
##          Pos Pred Value : 0.8925
##          Neg Pred Value : 0.5327
##              Prevalence : 0.6650
##          Detection Rate : 0.4150
##    Detection Prevalence : 0.4650
##       Balanced Accuracy : 0.7374
##
##        'Positive' Class : 0
##
```

```r
# ROC  Curve
r_pred = prediction(test$prob,test$V25)
r_roc = performance(r_pred, measure='tpr', x.measure='fpr')

plot(r_roc, main="ROC curve",lwd=2, col="blue",alpha=.6, colorize=F, xaxs="i", yaxs="i")
abline(a=0, b= 1)
```


ROC curve
True positive rate
False positive rate
0.0
0.2
0.4
0.6
0.8
1.0